I. Principal Component Analysis: real data

1. For the given data, de-mean each entry, using column sample means. Perform a PCA on the demeaned data. Clearly explain how you performed the PCA (submit pseudocode for your own PCA code, or reference and describe any function you called, i.e., from what library, how it works, what it takes as inputs and produces as outputs).

```
load ('PCA_comp1.mat');                        % Load raw data into OCTAVE
[X_norm, mu, sigma] = Normalization(X);        % First, normalize X (subtract mean & divided by sigma)
[U, S] = pca(X_norm);                          % Then, run PCA on dataset
```

```
% Normalization function: perform normalization to the input data
function [X_norm, mu, sigma] = Normalization(X)
% compute the column mean
mu = mean(X);
% subtract column mean from original matrix
X_norm = bsxfun(@minus, X, mu);
% compute sigma
sigma = std(X_norm);
% divide the normalized matrix by sigma
% X_norm = bsxfun(@rdivide, X_norm, sigma);
end
```

```
% pca function: perform PCA
function [U, S] = pca(X)
[m, n] = size(X);
Y = (1 / (m - 1)) * X' * X;            % Compute covariance matrix Y^T Y
[U, S, V] = svd(Y);                    % Perform SVD on covariance matrix
end
```

Pseudo-code:

1. Preprocessing: Subtract the *mean* and divided by *sigma*
2. Compute covariance matrix *SIGMA*
3. Compute 'eigenvectors' of matrix *SIGMA*
4. Select *k* appropriate principal components

2. State what portion of the variance in the data is contained in each of the 4 principal components. From these values, state how many true components are needed to represent the data.

```
proportion = zeros(size(S, 1), 1);
denominator = sum(sum(S));
```

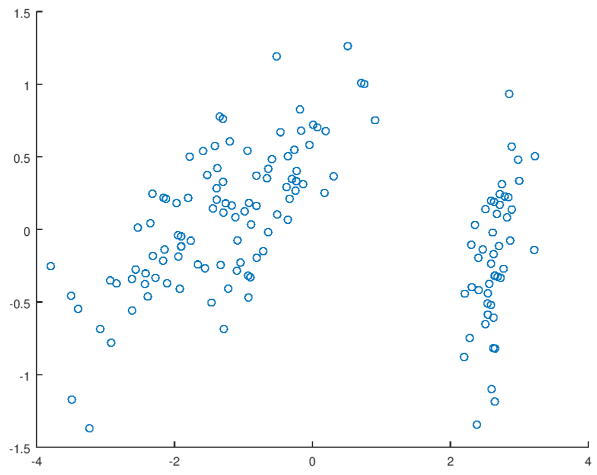
```
% compute the proportion of retained variance
for i = 1:size(S, 1)
        I = ones(i, 1);
        numerator = sum(S(1:i, 1:i) * I);
        proportion(i) = numerator / denominator;
end


% display the proportion matrix
disp(proportion);
```

```
Output:
       0.9246162    % 1st principal component variance retained proportion
       0.0530156    % 2nd principal component variance retained proportion
       0.0171851    % 3rd principal component variance retained proportion
       0.0051831    % 4th principal component variance retained proportion
```

Actually we should pick the smallest value of K (# of principal components) for which keeps 99% of variance retained. ***So in this case we only need 3 principal components to represent the data.***

3. On a 2D graph with the horizontal axis given by the first PC, and the vertical plot given by the second PC, plot the 2D representation of all the data points (i.e., find the 2D projection of each row). Discuss: (a.) are there any visually apparent clusters, if so, how many, and (b.) from this do you think you can conjecture anything about the number of species represented by the data?

```
Ureduce = U(:,1:2);              % reschedule the number of principal components
Z = X_norm * Ureduce;            % generate the projection Z
scatter(Z(:, 1), Z(:, 2));       % plot the scatter plot
```



(a.) There are 2 obvious clusters.
(b.) There are 2 separate species.

4. Repeat parts 1- 3 for *standardized* data: in addition to de-meaning the entries by column means, you should also scale by the inverse of the sample standard deviation, i.e., for each element x_{ij} in the original matrix X, normalize the element to $\tilde{x}_{ij}$ where

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{s_j}.$$

Note whether or not this scaling changes the outcome of your analysis.

```
load ('PCA_comp1.mat');                      % Load raw data into MATLAB/OCTAVE
[X_norm, mu, sigma] = Normalization(X);      % First, normalize X
[U, S] = pca(X_norm);                        % Run PCA
```

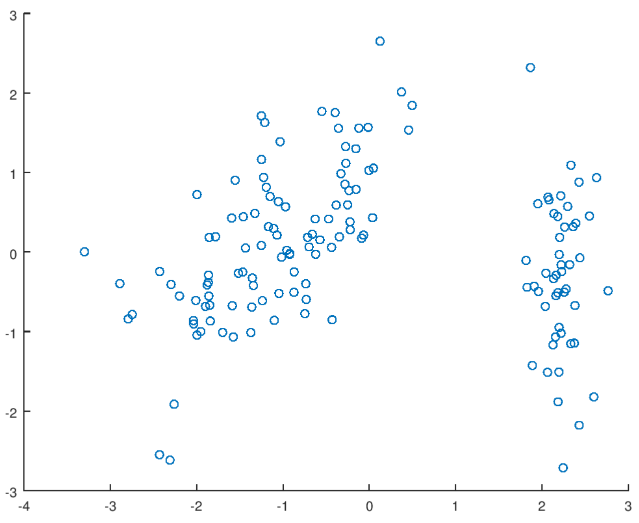
```
% Normalization function: perform normalization to the input data
function [X_norm, mu, sigma] = Normalization(X)
% compute the column mean
mu = mean(X);
% subtract column mean from original matrix
X_norm = bsxfun(@minus, X, mu);
% compute sigma
sigma = std(X_norm);
% divide the normalized matrix by sigma
X_norm = bsxfun(@rdivide, X_norm, sigma);
end
```

```
% pca function: perform PCA
function [U, S] = pca(X)
[m, n] = size(X);
Y = (1 / (m - 1)) * X' * X;                % Compute covariance matrix Y^TY
[U, S, V] = svd(Y);                        % Perform SVD on covariance matrix
end
```

```
proportion = zeros(size(S, 1), 1);
denominator = sum(sum(S));

% compute the proportion of retained variance
for i = 1:size(S, 1)
        I = ones(i, 1);
        numerator = S(i, i);
        proportion(i) = numerator / denominator;
end

% display the proportion matrix
disp(proportion);
```

```
Output:
      0.7277045      % 1 principal component variance retained proportion
      0.2303052      % 2 principal components variance retained proportion
      0.0368383      % 3 principal components variance retained proportion
      0.0051519      % 4 principal components variance retained proportion
```

```
Ureduce = U(:,1:2);            % reschedule the number of principal components
Z = X_norm * Ureduce;          % generate the projection Z
scatter(Z(:, 1), Z(:, 2));     % plot the scatter plot
```



(a.) There are 2 obvious clusters. (b.) There are 2 separate species.

5. Turn in a report including (a) a matrix with the 4 components for the data and their associated variances, and (b) plots for parts 3 and 4 (i.e., for both the demeaned and the standardized data sets).

(a)

```
disp(U);      % display the principal components
disp(S);       % display the eigenvalues which is their associated variance
% BELOW IS MATRIX GENERATED BY SCALED DATA
    -0.522372    -0.372318      0.721017      0.261996
      0.263355    -0.925556    -0.242033    -0.124135
    -0.581254    -0.021095    -0.140892    -0.801154
    -0.565611    -0.065416    -0.633801      0.523546

2.910818              0                    0                    0
       0         0.921221                  0                    0
       0              0              0.147353                   0
       0              0                    0              0.020608
```

```
% BELOW IS MATRIX GENERATED BY UNSCALED DATA
    -0.361590      -0.656540      0.580997      0.317255
      0.082269    -0.729712      -0.596418    -0.324094
    -0.856572       0.175767     -0.072524    -0.479719
    -0.358844       0.074706     -0.549061      0.751121

              4.224841              0                  0                0
                  0              0.242244              0                0
                  0                  0             0.078524             0
                  0                  0                 0            0.023683
```

(b) Plots see previous pages.

II. Regression analysis:

1. **Write a simple program** to compute a least squares solution for the case of linear or polynomial regression, and determine the lowest order model that fits the data reasonably well (order 1 is linear, and higher orders are polynomial).

Case 1: linear regression

```
% Load data to OCTAVE
load ('Comp1_IE529.mat');
X = lift_kg';            % 62x1 vector
y = putt_m';             % 62x1 vector

X = [ones(size(X, 1), 1) X];
theta = inv(X' * X) * X' * y

Output (ignored the sigma and r output)

beta =

    5.679442
    0.099479

residual =    94.583      (sum of squares of r)
```

Case 2: Polynomial Regression

```
% Load data to OCTAVE
load ('Comp1_IE529.mat');
X = lift_kg';            % 62x1 vector
y = putt_m';           % 62x1 vector
residual = zeros(15, 1);

X = [ones(size(X, 1), 1) X X .^ 2 X .^ 3];
Theta = inv(X' * X) * X' * y;
pred = X * Theta;
residual = (pred - y)' * (pred - y);
residual

Output:
residual = (sum of squares of r)
    51.08726      % order = 2
    44.57859      % order = 3
    44.47092       % order = 4 and so on….
    44.05494
    43.98594
    43.98366
```

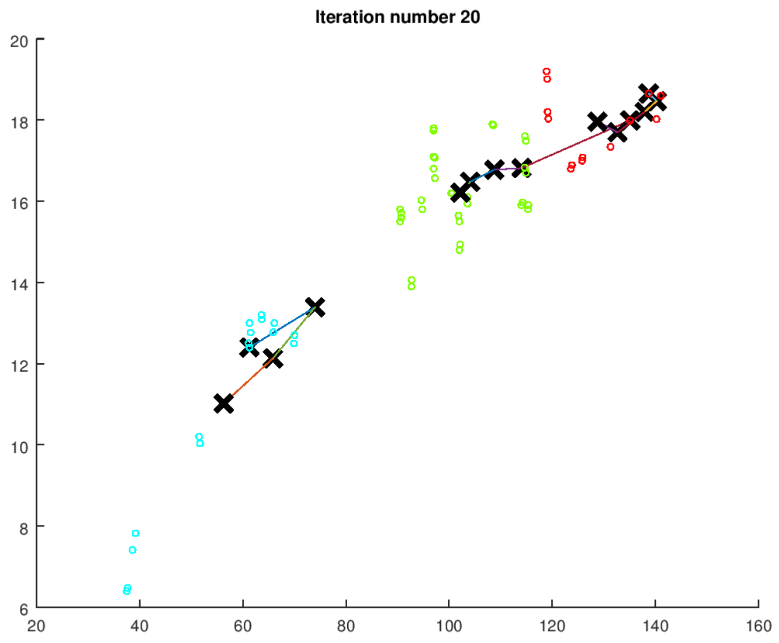
Based on the results of the residuals:
When it comes to linear regression, the residual is 94.583 which is highly large. When it comes to polynomial regression, when n=2, residual is 51.087 and n=3, residual is 44.579. However, when n=4, residual is 44.471 which has no much apparent differences compared to 44.579. In conclusion, I will select ***<u>Polynomial Regression and n=3.</u>***

2. Consider using an existing logistic regression function in MATLAB or Python to determine if a logistic model will fit the data much better or not. Discuss.

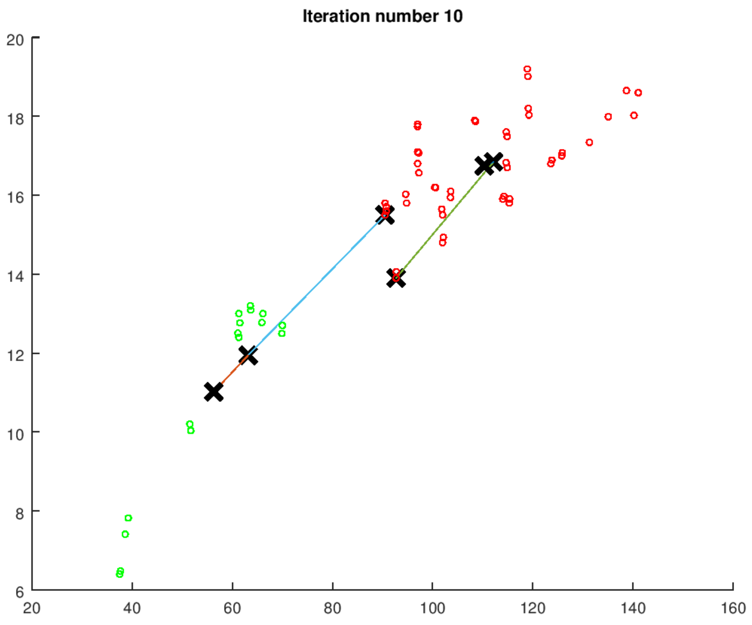
For logistic regression, we consider it as a binary classification, that is the Y of data is 0 or 1. In this case, this dataset is not appropriate for this regression.
Considering multinomial logistic regression, this dataset is also not suitable because this dataset consists continuous value not discrete categories. Even though multinomial logistic regression is used to predict categorical placement in or the probability of category membership on a dependent variable based on multiple independent variables.

Considering performing logistic regression based on the result of K-means.

Considering 3 clusters:



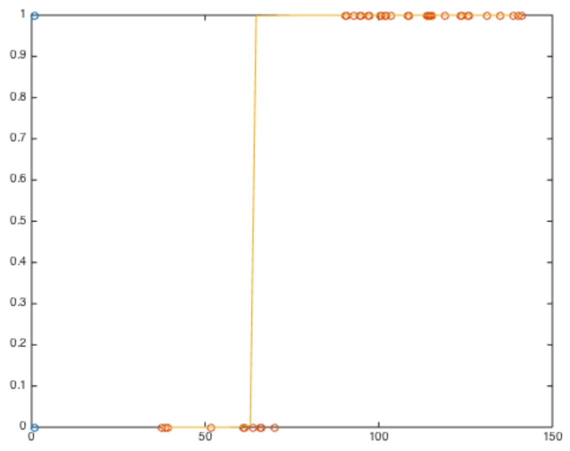
Considering 2 clusters:



```
% Load an example dataset that we will be using
load ('Comp1_IE529.mat');
x = lift_kg';              % 62x1 vector
y = putt_m';             % 62x1 vector

X(:, 1) = x;
X(:, 2) = y;

max_iters = 10;

% Select an initial set of centroids
K = 2;         % 2 Centroids

% Find the closest centroids for the examples using the
% initial_centroids
idx = findClosestCentroids(X, initial_centroids);

initial_centroids = kMeansInitCentroids(X, K);

%     Compute means based on the closest centroids found in the previous part.
centroids = computeCentroids(X, idx, K);

% Run K-Means algorithm. The 'true' at the end tells our function to plot
% the progress of K-Means
[centroids, idx] = runkMeans(X, initial_centroids, max_iters, true);
centroids
```

```
centroids =

       110.298          16.743
         56.285          11.017

% Load data
load ('Comp1_IE529.mat');
x = lift_kg';                % 62x1 vector
y = putt_m';              % 62x1 vector

[x_sort, x_idx] = sort(x);
y_sort = y(x_idx);

X1 = x_sort(1:sum(x_sort <= 80));
Y1 = zeros(size(X1, 1), 1);

X2 = x_sort(size(X1, 1) + 1:end);
Y2 = ones(size(X2, 1), 1);

y = [Y1; Y2];

% % Add intercept term to x and X_test
X = [ones(size(x_sort, 1), 1) x_sort];
initial_theta = zeros(n + 1, 1);

options = optimset('GradObj', 'on', 'MaxIter', 400);

%       Run fminunc to obtain the optimal theta
%       This function will return theta and the cost
[theta, cost] = ...
        fminunc(@(t)(costFunction(t, X, y)), initial_theta, options);
theta
cost

pred = sigmoid(X * theta);
xx = linspace(min(X(:,2)), max(X(:,2)), 62);
plot(X, y, 'o', xx, pred, '-')

theta =

       -49.32699
         0.60995
```

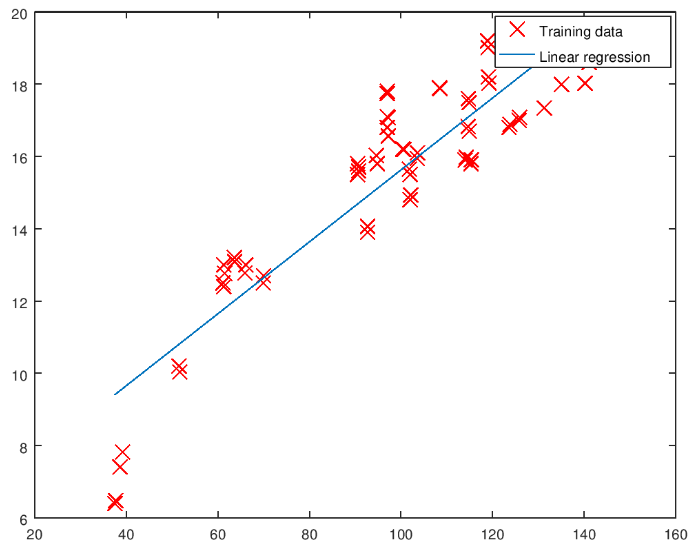
```
cost =        2.5143e-04
```



3. State which of your candidate models best describes the data, taking into account that simplicity is preferred. **Provide plots** of a linear fit, one polynomial fit (i.e., second order or higher) and if possible one logistic fit. Compute the sum-of-square of residuals, i.e., give the cost $\|\epsilon_i\|_2^2$, for each of the models plotted.

Case 1: Linear Regression

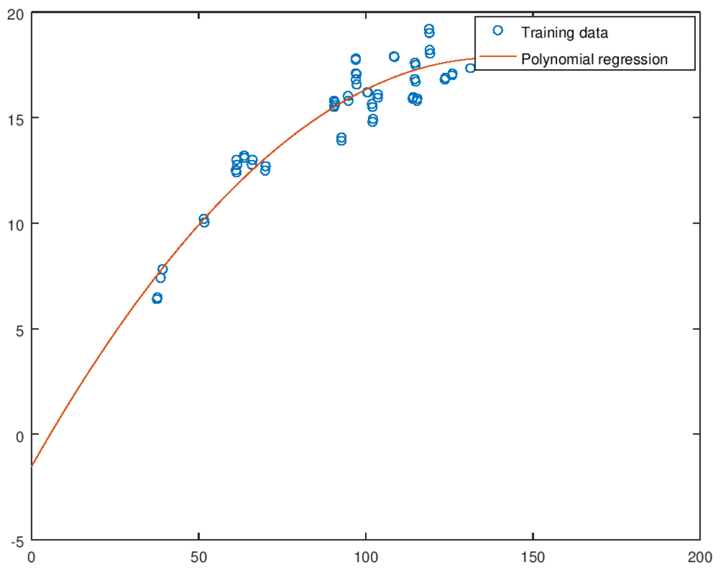
```
plotDataLR(X, y);
% Plot the linear fit
hold on;                                     % keep previous plot visible
plot(X, [ones(size(X, 1), 1) X] * beta, '-')
legend('Training data', 'Linear regression')
hold off                                     % don't overlay any more plots on this figure
% ------------------------------------------------------------------------------------------ %
function plotDataLR(x, y)
figure;                                      % open a new figure window
load ('Comp1_IE529.mat');
X = lift_kg';                                % 62x1 vector
y = putt_m';                                 % 62x1 vector
m = length(y);
plot(x, y, 'rx', 'MarkerSize', 10);
end
```



residual = 94.583

Case 2: Polynomial Regression

```
% Load data
load ('Comp1_IE529.mat');
X = lift_kg';           % 62x1 vector
y = putt_m';            % 62x1 vector

p = polyfit(X, y, 2);
pred = polyval(p,X);

residual = [pred - y]' * [pred - y];

t2 = 0:0.1:max(X);
y2 = polyval(p,t2);
plot(X,y,'o',t2,y2)

legend('Training data', 'Polynomial regression');
```



residual = 51.087